Solución: Cuaderno de ejercicios: Refactorización de código

Ejercicio 1: Refactos

Realiza la refactorización del siguiente programa

Solución:

```
public class Ejemplo1 {
   private static float cuadrado(float num) {
       return num * num;
   }

   public static float algoritmoX(float b) {
       float a = cuadrado(b) + 1;

       for(int i=0;i<3;i++) {
           b = cuadrado(b);
       }
       a += b;
       return a;

   }
   (String[] args) {
       System.out.println(algoritmoX(4));
       System.out.println(algoritmoX(-4));
       System.out.println(algoritmoX(0));
   }
}
```

Ejercicio 2: Empresa

Realiza la refactorización del siguiente programa

Solución:

```
public class Empresa {
   public String state;
   public final static String VIGO = "vigo";
   public final static String CORUNA = "coruna";
   public final static String LUGO = "lugo";

   public final static double VIGO_IMPUESTO = 0.21;
   public final static double CORUNA_IMPUESTO = 0.18;
   public final static double LUGO_IMPUESTO = 0.16;
   public final static double DEFAULT_IMPUESTO = 1;

   public final static double CORUNA_PUNTOS = 2;
   public final static double DEFAULT_PUNTOS = 0;

   public double base = 0.2;

   public double calc;
   public double points;

   private double calcularRate(String state) {
       if (state == VIGO) {
           return VIGO_IMPUESTO;
       } else if(state == LUGO) {
           return LUGO_IMPUESTO;
       }else if(state == CORUNA) {
           return CORUNA_IMPUESTO;
       } else {
           return DEFAULT_IMPUESTO;
       }
   }
   private double calcularPuntos(String state) {
       return state==CORUNA ? CORUNA_PUNTOS : DEFAULT_PUNTOS;
   }
   public double calculo() {
       double rate, amt=0;
       final double VALOR_CONSTANTE = 1.05;
       rate = calcularRate(state);
       points = calcularPuntos(state);

       amt = base * rate;
       calc = 2 * basis(amt) + extra(amt) * VALOR_CONSTANTE;
       return calc;
   }

   private double basis(double e) {
       return e + e;
   }

   private double extra(double e) {
       return e * e;
```

```
    }

    public static void main(String[] args) {
        Empresa e1 = new Empresa();
        e1.state = Empresa.LUGO;
        e1.calculo();
        System.out.println("c=" + e1.calc);
        System.out.println("c=" + e1.points);

        Empresa e2 = new Empresa();
        e2.state = Empresa.CORUNA;
        e2.calculo();
        System.out.println("c=" + e2.calc);
        System.out.println("c=" + e2.points);

        Empresa e3 = new Empresa();
        e3.state = Empresa.VIGO;
        e3.calculo();
        System.out.println("c=" + e3.calc);
        System.out.println("c=" + e3.points);
    }
}
```

Ejercicio 3: Shape

Realiza la refactorización del siguiente programa

Solución:

```
package refactorizacion2;

public class Shape {
   public String color;
   public void colorDefecto() {
       color = "negro";
   }

   public static void main(String[] arg) {
       Square cuadrado = new Square(20);
       System.out.println (cuadrado.area() );
   }
}

class Circle extends Shape{
   public double radio;
   public double area() {
       return Math.PI * radio * radio;
   }
   public Circle(double radio) {
       this.radio = radio;
   }

}

class Square extends Shape{
   public double lado;
   public double area() {
       return lado * lado;
   }
   public Square(double lado) {
       this.lado = lado;
   }

}

class RightTriangle extends Shape{
   public double base;
   private double altura;
   public double area() {
       return base * altura / 2;
   }
   public RightTriangle(double base, double altura) {
       this.base = base;
       this.altura = altura;
   }

}
```

Ejercicio 4: Alumno

Realiza la refactorización del siguiente programa
Realiza también los test unitarios

Solución: Refactorización

```java
/**
* Clase que permite calcular la calificacion del alumno en funci'n de sus retrasos
* @author Angel
* @version 1.0
* @since 11/02/2020
*/
public class Alumno {

   /**
    * Determina el numero maximo de faltas que se permiten a un alumno
    */
   public static final int MAX_NUM_FALTAS = 5;
   /**
    * Valor de la nota cuando el alumno no ha cometido las faltas suficientes
    */
   public static final int NOTA_SIN_FALTAS = 5;
   /**
    * Valor de la nota cuando el alumno se ha pasado del n' de faltas
    */
   public static final int NOTA_CON_FALTAS = 1;

   /**
    * Almacena la cantidad de faltas del alumno
    */
   private int numeroDeRetrasos;

   /**
    * Contructor del alumno a partir de sus faltas
    * @param numeroDeRetrasos Las faltas del alumno
    */
   public Alumno(int numeroDeRetrasos) {
       this.numeroDeRetrasos = numeroDeRetrasos;
   }

   /**
    * Devuelve la cantidad de faltas cometidas por el alumno
    * @return la cantidad de faltas
    */
   public int getNumeroDeRetrasos() {
       return numeroDeRetrasos;
   }

   /**
    * Establece el n'mero de faltas cometidas por el alumno
    * @param numeroDeRetrasos la cantidad de faltas
    */
   public void setNumeroDeRetrasos(int numeroDeRetrasos) {
       if(numeroDeRetrasos < 0  ||  numeroDeRetrasos > 300) {
           throw new NumberFormatException();
```

```
        }
        this.numeroDeRetrasos = numeroDeRetrasos;
    }

    /**
     * Devuelve la calificaci'n del alumno en funci'n de sus faltas
     * @return la calificaci'n
     */
    public int obtenerCalificacion() {
        return masDeCincoRetrasos() ? NOTA_CON_FALTAS : NOTA_SIN_FALTAS;
    }

    /**
     * Determina si las faltas superar el maximo permitido
     * @return un booleano indicando si se superan el maximo de faltas
     */
    private boolean masDeCincoRetrasos() {
        return numeroDeRetrasos > MAX_NUM_FALTAS;
    }


    public static void main(String[] args) {
        Alumno a1 = new Alumno(5);
        System.out.println(a1.obtenerCalificacion());

        Alumno a2 = new Alumno(1);
        System.out.println(a2.obtenerCalificacion());

        Alumno a3 = new Alumno(7);
        System.out.println(a3.obtenerCalificacion());
    }

}
```

Solución: Test

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class AlumnoTest {
    @Test
    void test() {
        Alumno a = new Alumno(12);
        int c = a.obtenerCalificacion();
        assertEquals(1, c);

        a = new Alumno(2);
        c = a.obtenerCalificacion();
        assertEquals(5, c);
    }

}
```

Ejercicio 5: GestorBD

Realiza la refactorización del siguiente programa
Solución:

```
import java.sql.Connection;
import java.util.ArrayList;
import java.util.Date;

public class Estudiante {
   String nombre;
   String telefono;
   Date fechaNac;

   GestorBD gestorBD = new GestorBD();

   public ArrayList getListaEstudiantes() {
       Connection c = gestorBD.getConnection();
       // obtenemos la lista de alumnos de la base de datos (* imag'natelo)
       // ... select * from estudiantes
       gestorBD.cerrarConnection();
       return null;
   }


   public void borrarEstudiantes() {
       Connection c = gestorBD.getConnection();
       // borramos los alumnos de la base de datos (* imag'natelo)
       // ... delete from estudiantes

       gestorBD.cerrarConnection();
   }
}
```

```
import java.sql.Connection;
import java.util.ArrayList;

public class Profesor {
   String nombre;
   String telefono;
   float nonima;

   GestorBD gestorBD = new GestorBD();


   public ArrayList getListaProfesores() {
       Connection c = gestorBD.getConnection();
       // obtenemos la lista de profesores de la base de datos (* imag'natelo)
       // ... select * from profesores
       gestorBD.cerrarConnection();
       return null;
   }
}
```

```java
import java.sql.Connection;

public class GestorBD {

   // Quizas estos 2 metodos getConnection y cerrarConection
   // Deber'an ser static
   public Connection getConnection() {
       // Abrimos la conexi'n con la base de datos Ej MySQL (* imag'natelo)
       return null;
   }

   public void cerrarConnection() {
       // Cerramos la conexi'n con la base de datos (* imag'natelo)
   }
}
```

Ejercicio 6: Minecraft Cubos

Realiza la refactorización del siguiente programa
Solución:

```
import java.awt.geom.Point2D;
abstract class Cubo {
   String nombre;
   String textura;
   String danno; //TOFIX: usar un entero en lugar de un string.
   Point2D posicion;
   public Cubo() {
       posicion = new Point2D.Float(0, 0);
   }
   void dibujar() {
       // dibujar en pantalla un cubo con textura = textura
   }
   public static void main(String[] args) {
       Cubo[] arrCubos = new Cubo[100];
       arrCubos[0] = new CuboTNT();
       arrCubos[1] = new CuboTNT();
       arrCubos[2] = new CuboHierba();
       arrCubos[3] = new CuboMetal();

       System.out.println(arrCubos[2].nombre);
   }
}

class CuboTNT extends Cubo {
   int rangoExplosion = 200;
   int dannoExplosion = 10;
   public CuboTNT(){
       super.nombre = "tnt";
       super.danno = "10";
       super.textura = "tnt.png";
   }
   public void explotar() {
       // Buscar usuario en un rango de 200 (rangoExplosion)
       // Aplicarles un da'o de 10/200 = da'o/rangoExplosion
   }
}

class CuboHierba extends Cubo {
   public CuboHierba(){
       super.nombre = "hierba";
       super.danno = "100";
       super.textura = "hierba.png";
   }
}

class CuboMetal extends Cubo {
   public CuboMetal(){
       super.nombre = "metal";
       super.danno = "1000";
       super.textura = "metal.png";
   }
}
```

Ejercicio 7: Animales

Realiza la refactorización los siguientes métodos

Solución:

```
abstract class Animal{
   int peligrosidad;
   abstract void matar();
}

class Leon extends Animal{
   public Leon() { super.peligrosidad = 10; }
   @Override void matar() {
       //disparo de ca'on
   }
}
class Lobo extends Animal{
   public Lobo() { super.peligrosidad = 8; }
   @Override void matar() {
       //disparo bala plata
   }
}
class Gato extends Animal{
   public Gato() { super.peligrosidad = 1; }
   @Override void matar() {
       //patada
   }
}


public class Principal {
   public static void main(String[] args) {
       sumaA(3,5,7,10,22);
       sumaC( new int[] {3,5,7} );
   }
   public char obtenerPrimerCaracterEnMinusculas(String texto) {
       return texto.toLowerCase().charAt(0);
   }

   /*public int sumaO(int a, int b, int c, int d, int e, int f) {
       return a + b + c + d + e + f;
   }
   static public int sumaC(int[] datos) {
       int suma = 0;
       for(int i=0;i<datos.length;i++) {
           suma += datos[i];
       }
       return suma;
   }*/
   static public int sumaA(int... datos) {
       int suma = 0;
       for(int i=0;i<datos.length;i++) {
           suma += datos[i];
       }
       return suma;
```

```
    }

    private int obtenerNumeroCaracteres(String texto) {
        return texto.length();
    }

    public int calcularPeligrosidad(Animal animal) {
        return animal.peligrosidad;
    }

    public char otenerUltimoCaracterEnMinusculasYSinEspacios(String texto) throws
Exception {
        if (texto == null) throw new Exception("Error, el texto no puede ser nulo");

        return texto.trim().toLowerCase().charAt(texto.length() - 1);
    }
}
```

Ejercicio 8: Cliente

Realiza la refactorización del siguiente programa

Solución:

```
class Direccion {
	public String direccion;
	public String ciudad;
	public int cp;
	final static private String[] PROVINCIAS = {"", "Alava", "Albacetge", "Alicante",
"Almería"};

	public boolean validarDireccion() {
		// validamos los datos del cliente
		return true;
	}

	public String getProvincia(int cp) {
		int codigoProvincia = cp / 100;
		return PROVINCIAS[codigoProvincia];
	}
}

public class Cliente {
	public String nombre;
	public String telefono;
	public String email;
	public Direccion direccion;

	public void registrarCliente() {
		// guardamos el cliente
	}
}
```

Ejercicio 9: Venta

Realiza la refactorización del siguiente programa.
Haz los test unitaros

Solución: refactorización
import org.hamcrest.core.IsInstanceOf;

// Documentar, test, refactorizar y optimizar

```
abstract class Cliente{
	/** Nombre del cliente que realiza la compra */
	String nombre;
	/** Determina si un cliente es tambien empleado */
	boolean esEmpleado;
	/** Determina si el cliente posee cupón gratuito */
	boolean tieneCuponGratis;
}
class ClienteGold extends Cliente{ }
class ClientePlatinum extends Cliente{ }
class ClienteNormal extends Cliente{ }
/**
 * Clase que permite calcular el coste de una venta
 * @author Angel
 * @version 1.0
 *
 */
public class Venta {
	/** coste de la venta */
	float coste;
	/** descuento de la venta (en porcentaje con decimales) */
	float dto;

	Cliente cliente;

	private boolean isFree() {
		return cliente.esEmpleado ||
			cliente instanceof ClienteGold ||
			cliente instanceof ClientePlatinum  ||
			cliente.tieneCuponGratis;
	}

	/**
	 * Método que calcula el total con descuento de la venta
	 * @return El valor del total con descuento
	 */
```

```
	float calculaCosteTotal() {
		//if(isFree()) return 0;
		//return coste *(1-dto);
		return isFree() ? 0 : coste *(1-dto);
	}
	/* Ampliación:
	 * Como los precios son muy altos, algunos clientes prefieren
	 * pagar el producto dentro de unos meses
	 * Crea una función que permita calcular el coste total
	 * - Especificando la cantidad de meses y
	 * - la tasa de interés que se le va a aplicar
	 *  (usa la formula del interés simple, googlea!)
	 * */

	/**
	 * Calcula el coste total con intereses
	 * @param meses tiempo en meses
	 * @param tasa tasa a aplicar
	 * @return el coste con intereses
	 */
	float calcularCosteConIntereses(int meses, float tasa) {
		return calculaCosteTotal() * tasa * meses + 1;
	}
}
```

Solución: Test

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

class VentaTest {

	@Test
	void testCalcula_coste_totalConCuponGratis() {
		Venta v = new Venta();
		v.cliente = new ClienteNormal();
		v.cliente.tieneCuponGratis = true;
		v.coste=4000;
		v.dto=5;
		float total = v.calculaCosteTotal();
		assertEquals(0, total);
	}
	@Test
```

```
	void testCalculaCosteDeVentaNormal() {
		Venta v = new Venta();
		//v.cliente.tipoDeCliente = Cliente.TipoCliente.NORMAL;
		v.cliente = new ClienteNormal();
		v.cliente.tieneCuponGratis=false;
		v.coste=1000;
		v.dto=0.5f;
		float total = v.calculaCosteTotal();
		assertEquals(500, total);
	}
	@Test
	void testCalculaCosteEsEmpleado() {
		Venta v = new Venta();
		v.cliente = new ClienteNormal();
		v.cliente.esEmpleado = true;
		v.coste=1000;
		v.dto=0.5f;
		float total = v.calculaCosteTotal();
		assertEquals(0, total);
	}

}
```

Ejercicio 10: Factura

Realiza la refactorización del siguiente programa
Solución:

```
import java.util.Date;

class Cliente {
   String nombreCliente;
   String nifCliente;
   String dirCliente;
}

class Empresa {
   String cifEmpresa;
   String dirEmpresa;
}

class Articulo {
   String nombre;
   float precio;
}

public class Factura {
   String Numero;
   Date vencimiento;
   Articulo[] articulos;
   Empresa empresa;
   Cliente cliente;

   void imprime_factura_por_consola() {
       System.out.println(getDocumentoFactura());
   }

   void genearPdf() {
       // generadorPdf(getDocumentoFactura());
   }

   void enviarEmail() {
       // mail("gonzalezm.angel@gmail.com", "Envio factura", getDocumentoFactura())
   }

   StringBuilder getCabeceraDocumentoFactura(StringBuilder texto) {
       // Pintamos la cabecera
       texto.append("FACTURA n' " + Numero);
       texto.append("\n");
       texto.append("Fecha: " + new Date());
       texto.append("\n");
       return texto;
   }

   StringBuilder getDatosEmpresaDocumentoFactura(StringBuilder texto) {
       // Pintamos los datos de la empresa que genera la factura
       texto.append("CEBEM SL");
       texto.append("\n");
       texto.append("CIF: " + empresa.cifEmpresa);
       texto.append("\n");
```

```
        texto.append("Direcci'n" + empresa.dirEmpresa);
        texto.append("\n");
        texto.append("36204 Vigo (Pontevedra)");
        texto.append("\n");
        return texto;
    }

    StringBuilder getDatosClienteDocumentoFactura(StringBuilder texto) {

        // Pintamos los datos del cliente
        texto.append("CLIENTE:");
        texto.append("\n");
        texto.append("-------------------------");
        texto.append("\n");
        texto.append("Nombre:" + cliente.nombreCliente);
        texto.append("\n");
        texto.append("NIF: " + cliente.nifCliente);
        texto.append("\n");
        texto.append("Direcci'n" + cliente.dirCliente);
        texto.append("\n");
        texto.append("36204 Vigo (Pontevedra)");
        texto.append("\n");
        return texto;
    }

    StringBuilder getDatosArticulosDocumentoFactura(StringBuilder texto) {
        // Pintamos los datos de la factura
        texto.append("ARTICULOS:");
        texto.append("\n");
        texto.append("-------------------------");
        texto.append("\n");
        float total = 0;
        for (int i = 0; i < articulos.length; i++) {
            texto.append(articulos[i].nombre + "      " + articulos[i].precio);
            total = total + articulos[i].precio;
        }
        texto.append("               TOTAL:        " + total);
        texto.append("\n");
        return texto;

    }

    String getDocumentoFactura() {
        StringBuilder texto = new StringBuilder();

        texto = getCabeceraDocumentoFactura(texto);
        texto = getDatosEmpresaDocumentoFactura(texto);
        texto = getDatosClienteDocumentoFactura(texto);
        texto = getDatosArticulosDocumentoFactura(texto);

        // Pintamos el pie de la factura
        texto.append("Vencimiento: " + vencimiento.toString());
        return texto.toString();

    }
}
```

Ejercicio 11: Proveedor

Realiza la refactorización del siguiente programa
Solución:

```
class Proveedor {
   boolean activo = true;
}

public abstract class OrdenVenta {
   float total;
   Proveedor proveedor;
   float peso;
   public abstract float getDescuento();
   public float calculaDescuento() {
       if (proveedor.activo == true && peso < 1000 && this instanceof OrdenVentaNormal)
{
           return getDescuento();
       } else {
           return getDescuento();
       }
   }

   class OrdenVentaNormal extends OrdenVenta {
       public final static float DELTA = 0.5f;
       public float getDescuento() {
           float coste = total * DELTA;
           if (coste > 100) coste = 100;
           return coste;
       }
   }

   class OrdenVentaEspecial extends OrdenVenta {
       public final static float DELTA = 0.8f;
       public float getDescuento() {
           float coste = total * DELTA;
           if (coste > 200) coste = 200;
           return coste;
       }
   }

}
```

Ejercicio 12: AngryBirds

Crea las clases (métodos y atributos) del juego AngryBirds
Solución:

```
import java.awt.Color;

abstract class Entity {
   String sprite;
   float tamaño;
   float peso;
   float velocidad;
   float aceleracion;
   float daño;

   abstract void mover();

}

interface Lanzable {
   public void lanzar();
}

interface Colisionable {
   public void getBounds();

   public void onCollision();
}

abstract class Pajaro extends Entity implements Lanzable, Colisionable {
   public void lanzar() {
   }
}

class PajaroRojo extends Pajaro {
   void mover() {
   }

   public void getBounds() {
   }

   public void onCollision() {
   }
}

class PajaroNegro extends Pajaro {
   void mover() {
   }

   public void getBounds() {
   }

   public void onCollision() {
   }
}

class PajaroGranate extends Pajaro {
```

```
    void mover() {
    }

    public void getBounds() {
    }

    public void onCollision() {
    }
}

abstract class Cerdo extends Entity implements Colisionable {

}

class CerdoRey extends Cerdo {
    void mover() {
    }

    public void getBounds() {
    }

    public void onCollision() {
    }
}

abstract class Estructura extends Entity implements Colisionable {

}

class CajaMetal extends Estructura {
    void mover() {
    }

    public void getBounds() {
    }

    public void onCollision() {
    }
}

class Terreno {

}

class Tirachinas {
    float fuerza;
    float angulo;
    String sprite;

    void lanzar(Lanzable cosaALanzar) {

    }
}

class Item extends Entity implements Colisionable {
    void mover() {
```

```
    }

    public void getBounds() {
    }

    public void onCollision() {
    }
}

public class Juego {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Ejercicio 13: Validación de datos

Crea las clases (métodos y atributos) del juego AngryBirds
Solución:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class DataValidation {

    public boolean validarEmail(String email) throws Exception {
        if (email == null) {
            throw new Exception("Email no puede ser nulo");
        }
        Stringn regexEmail = "^[\\w-_\\.+]*[\\w-_\\.]\\@([\\w]+\\.)+[\\w]+[\\w]$";
        Pattern pattern = Pattern.compile(regexEmail);
        Matcher matcher = pattern.matcher(email);
        return matcher.matches();
    }

    /*
     * Este método valida un codigo postal.
     * Es decir un numero entre 01000 y 52999:
     */
    public boolean validarCP(String input) {
        /*
         * return input.length() == 5
         * && Integer.valueOf(input) >= 1000
         * && Integer.valueOf(input) <= 52999;
         */

        Stringn regexCP = "^(?:0?[1-9]|[1-4]\\d|5[0-2])\\d{3}$";
        Pattern pattern = Pattern.compile(regexCP);
        Matcher matcher = pattern.matcher(input);
        return matcher.matches();
```

```
    }

    boolean validarGenerico(String texto, String regex) {
        Pattern pattern = Pattern.compile(regex);
        Matcher matcher = pattern.matcher(texto);
        return matcher.matches();
    }
}
```

Ejercicio 14: Creador de usuarios

Realiza la refactorización del siguiente programa
Solución:

```
//alta cohesion bajo acoplamiento
class User {
   User(String nombre, String pass) {
   }
   String nombre;
   String email;
   String password;
}

class Validator {

        static boolean valid(String valor) {
           return valor != null && valor.length() > 0;
        }

        static boolean validateUser(String email, String pass) throws Exception{
                if (!valid(email)) throw new Exception("Email incorrecto");
             if (!valid(pass)) throw new Exception("Password incorrecto");
             return true;
        }
}
public class UserCreator {
   User create(String email, String pass) throws Exception  {
           Validator.validateUser(email, pass);
        return new User(email, pass);
   }
}
```

Ejercicio 15: Videoclub

Realiza la refactorización del siguiente programa
Solución:

```java
import java.time.LocalDate;
import static java.time.temporal.ChronoUnit.DAYS;

abstract class Pelicula {
   float precio;
   String titulo;
   String sinopsis;
   abstract float getPrecio();
}

class PeliculaNormal extends Pelicula{
       float getPrecio(){ return precio*1; }
}
class PeliculaEstreno extends Pelicula{
       float getPrecio(){ return precio*2; }
}
class PeliculaClasica extends Pelicula{
       float getPrecio(){ return precio*3; }
}

class Alquiler {
   Pelicula pelicula;
   LocalDate fechaAlquiler = LocalDate.now();
   final static int MIN_DIAS_ALQUILER = 7;
   final static int MAX_DIAS_ALQUILER = 20;
   final static int RECARGO = 10;
   final static int PENALIZACION = 250;

   long getDiasAlquiler() {
       LocalDate fechaActual = LocalDate.now(); // fecha actual
       return DAYS.between(fechaAlquiler, fechaActual);
   }

   private float calcularRecargo() {
          if (getDiasAlquiler() <= MIN_DIAS_ALQUILER) return 0;
          if (getDiasAlquiler() >= MIN_DIAS_ALQUILER+1
                       && getDiasAlquiler() < MAX_DIAS_ALQUILER) return
getDiasAlquiler() * RECARGO;
       if (getDiasAlquiler() > MAX_DIAS_ALQUILER) return PENALIZACION;
          return 0;
   }

   float calculaTotalAlquiler() {
       return calcularRecargo() + pelicula.getPrecio();
   }
}


class GestionAlquieres{
       static float calcularTotalAlquileres(Alquiler[] alquileres) {
              float total = 0;
              for (Alquiler al : alquileres) {
                  total += al.calculaTotalAlquiler();
              }
              return total;
          }
```

```
        static boolean esMoroso(Alquiler[] alquileres) {
           for (Alquiler al : alquileres) {
                   if (al.getDiasAlquiler() > Alquiler.MAX_DIAS_ALQUILER) return true;
           }
            return false;
        }
}

class Direccion{
        String calle;
        String piso;
        String letra;
        String cp;
}

class Cliente {
   String nombre;
   String telefono;
   Direccion direccion;

   Alquiler[] alquileres;
}



public class VideoClub {

}
```

Ejercicio 16: Sanitarios

Realiza la refactorización del siguiente programa

Solución:

```
class Enfermero extends Sanitario {
   private int inyeccionesSuministradas;

   public Enfermero(Sanitario jefe, int sueldoBruto, int inyeccionesSuministradas) {
       super(jefe, sueldoBruto);
       this.inyeccionesSuministradas = inyeccionesSuministradas;
   }
}

class Medico extends Sanitario {
   private int numeroGuardias;
```

```
   private Sanitario[] sanitariosACargo;

   public Medico(Sanitario jefe, int sueldoBruto, int numeroGuardias, Sanitario[]
sanitariosACargo) {
       super(jefe, sueldoBruto);
       this.numeroGuardias = numeroGuardias;
       this.sanitariosACargo = sanitariosACargo;
   }
}

class MedicosFamilia extends Medico {

   public MedicosFamilia(Sanitario jefe, int sueldoBruto, int numeroGuardias,
Sanitario[] sanitariosACargo) {
       super(jefe, sueldoBruto, numeroGuardias, sanitariosACargo);
   }
}

class Cirujanos extends Medico {
   private int numeroOperaciones;

   public Cirujanos(Sanitario jefe, int sueldo_bruto, int numeroGuardias, Sanitario[]
sanitariosACargo,
           int numeroOperaciones) {
       super(jefe, sueldo_bruto, numeroGuardias, sanitariosACargo);
       this.numeroOperaciones = numeroOperaciones;
   }

}

public class Sanitario {
   private Sanitario jefe;
   private int sueldoBruto;

   public Sanitario(Sanitario jefe, int sueldo_bruto) {
       this.jefe = jefe;
       this.sueldoBruto = sueldo_bruto;
   }

}
```

Ejercicio 17: Laberinto

Realiza la refactorización del siguiente programa

El siguiente programa, resuelve un laberinto, indicando en que dirección tienes que moverte para llegar a la salida:
Refactorízalo.

Solución:

```java
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.LinkedList;

class Maze {
   public int[][] maze;
   public LinkedList<Position> path = new LinkedList<Position>();
   public Position start;
}

class Position {
   public int x;
   public int y;

   public Position(int y, int x) {
       this.x = x;
       this.y = y;
   }
}

public class MazeSolver {
   // 0 = wall
   // 1 = path
   // 2 = destination

   private static int[][] maze = {
           { 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 },
           { 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0 },
           { 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1 },
           { 1, 1, 1, 2, 0, 1, 0, 1, 0, 1, 0 },
           { 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0 },
           { 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1 }
   };

   public void setMaze(int[][] maze) {
       this.maze = maze;
   }

   public void randomMaze() {

   }

   static LinkedList<Position> path = new LinkedList<Position>();

   public static void main(String[] args) throws FileNotFoundException {
       solveMaze();
       for (Position p : path) {
           System.out.println(p.x + " " + p.y);
       }

   }

   public static String[] solveMaze() throws FileNotFoundException {
       ArrayList<String> movimientos = new ArrayList<String>();
```

```
        Position p = new Position(4, 8);
        path.push(p);

        while (true) {
            int y = path.peek().y;
            int x = path.peek().x;

            maze[y][x] = 0;

            // down
            if (isValid(y + 1, x)) {
                if (maze[y + 1][x] == 2) {
                    // System.out.println("Moved down. You won");
                    movimientos.add("Moved down. You won");
                    return (String[]) movimientos.toArray();
                } else if (maze[y + 1][x] == 1) {
                    // System.out.println("Moved down");
                    movimientos.add("Moved down");
                    path.push(new Position(y + 1, x));
                    continue;
                }
            }

            // left
            if (isValid(y, x - 1)) {
                if (maze[y][x - 1] == 2) {
                    movimientos.add("Moved left. You won");
                    return (String[]) movimientos.toArray();
                } else if (maze[y][x - 1] == 1) {
                    movimientos.add("Moved left");
                    path.push(new Position(y, x - 1));
                    continue;
                }
            }

            // up
            if (isValid(y - 1, x)) {
                if (maze[y - 1][x] == 2) {
                    movimientos.add("Moved up. You won");
                    return (String[]) movimientos.toArray();
                } else if (maze[y - 1][x] == 1) {
                    movimientos.add("Moved up");
                    path.push(new Position(y - 1, x));
                    continue;
                }
            }

            // right
            if (isValid(y, x + 1)) {
                if (maze[y][x + 1] == 2) {
                    movimientos.add("Moved right. You won");
                    return (String[]) movimientos.toArray();
                } else if (maze[y][x + 1] == 1) {
                    movimientos.add("Moved right");
                    path.push(new Position(y, x + 1));
                    continue;
```

```
                }
            }

            path.pop();
            movimientos.add("Moved back");
            if (path.size() <= 0) {
                return (String[]) movimientos.toArray();
            }
        }

    }

    private static boolean isValid(int y, int x) {
        if (y < 0 ||
                y >= maze.length ||
                x < 0 ||
                x >= maze[y].length) {
            return false;
        }
        return true;
    }

}
```

Ejercicio 18: Carta

Dada la siguiente clase que modela el comportamiento de un famoso juego de cartas. Se pide refactorizar la misma para obtener un código más limpio.

Dificultad: ★ ★ ★

```
import java.util.ArrayList;

class Poder {
    String nombre;
    int valor;
}

interface Atacador {
    public float calcularDanno();
}

class CartaPlanta extends CartaOptimizada implements Atacador {
    public float calcularDanno() {
        if (estaCao())
            return 0;
```

```
        return danno >> 1;
    }
}

class CartaFuego extends CartaOptimizada implements Atacador {
    public float calcularDanno() {
        if (estaCao())
            return 0;
        return danno << 1;
    }
}

class CartaAgua extends CartaOptimizada implements Atacador {
    public float calcularDanno() {
        if (estaCao())
            return 0;
        return danno;
    }
}

class CartaElectrico extends CartaOptimizada implements Atacador {
    int voltaje = 5; // Cantidad de voltaje, usado para las cartas de tipo
Electrico

    public float calcularDanno() {
        if (estaCao())
            return 0;
        if (voltaje <= 0)
            return 0;
        return danno * voltaje * 0.1f;
    }

}

public abstract class CartaOptimizada {
    String nombre = "default"; // nombre de la carta
    String descripcion; // Breve descripción de la carta
    boolean bloquedada = false; // Determina si la carta está bloqueada
    int danno = 100; // Daño que produce la carta al atacar
    int ps = 10; // Puntos de vida

    // Poder[] poderes; // poderes de la carta
```

```
    ArrayList<Poder> poderes2 = new ArrayList<>(); // Otra forma mejor usando
arrayList

    boolean estaCao() {
        if (ps <= 0 || bloquedada)
            return true;
        return false;
    }

    public static void main(String[] args) {
        CartaElectrico c = new CartaElectrico();
        System.out.println(c.calcularDanno());
    }
}
```

Ejercicio 19: Encripta

Refactoriza el siguiente codigo fuente.
Si puedes optimizar algo, hazlo también.

```
class Encripta {
    public String encripta(String textoClaro) {
        String result = "";
        String letras = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        String conversion = "48()3=#-|1C7Wn06@_$+VUM%I>";

        textoClaro = textoClaro.toUpperCase();

        for (int i = 0; i < textoClaro.length(); i++) {
            char letraSeleccionada = textoClaro.charAt(i);
            int posicionLetra = letras.indexOf(letraSeleccionada);
            result += conversion.charAt(posicionLetra);
        }
        return result;
    }

    public static void main(String[] args) {
        Encripta e = new Encripta();
        System.out.println(e.encripta("sala"));
    }
```

```
}
```

Ejercicio 20: Vehículo

```
// Añade un nuevo vehiculo (moto)
// Refactoriza el código
interface Volable{
	public void volar();
}
abstract public class Vehiculo {
	String nombre;
	float potencia;
	float peso;

	float getConsumo() {
		return potencia * peso;
	}
}

class moto extends Vehiculo{
}

class avion extends Vehiculo implements Volable{
	public void volar() {
		//...
	}
}

class avioneta extends Vehiculo implements Volable{
	public void volar() {
		// ...
	}
}
```